Common Problems and Solutions

Problems with buildr-all-in-one-1.4.6 on Ubuntu 11.10

The buildr-all-in-one-1.4.6 gave me the error below on Ubuntu 11.10. Reinstalling libncursesw5 didn't help.
I had to create the following symlink to get it working:

```
sudo ln -s /lib/libncurses.so.5 /usr/lib/libncurses.so
```

```
./bin/buildr --version
LoadError: Could not open library 'ncursesw' : ncursesw: cannot open shared object file: No such file or
directory. Could not open library 'libncursesw.so' : libncursesw.so: cannot open shared object file: No such
file or directory. Could not open library 'libncursesw' : libncursesw: cannot open shared object file: No such
file or directory. Could not open library 'ncurses' : ncurses: cannot open shared object file: No such file or
directory. Could not open library 'libncurses.so' : /usr/lib/libncurses.so: file too short. Could not open
library 'libcurses' : libcurses: cannot open shared object file: No such file or directory. Could not open
library 'libcurses.so' : /usr/lib/libcurses.so: file too short
           ffi_lib at /home/alexis/buildr-1.4.6/lib/ruby/site_ruby/shared/ffi/library.rb:82
           collect at org/jruby/RubyArray.java:2336
           ffi_lib at /home/alexis/buildr-1.4.6/lib/ruby/site_ruby/shared/ffi/library.rb:64
           NCurses at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/ffi-ncurses-0.4.0/lib/ffi-ncurses.rb:64
               FFI at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/ffi-ncurses-0.4.0/lib/ffi-ncurses.rb:38
            (root) at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/ffi-ncurses-0.4.0/lib/ffi-ncurses.rb:37
           require at org/jruby/RubyKernel.java:1038
           require at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/ffi-ncurses-0.4.0/lib/ffi-ncurses.rb:59
  SystemExtensions at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/highline-1.6.2/lib/highline
/system_extensions.rb:90
          HighLine at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/highline-1.6.2/lib/highline
/system_extensions.rb:13
            (root) at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/highline-1.6.2/lib/highline
/system_extensions.rb:12
           require at org/jruby/RubyKernel.java:1038
           require at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/highline-1.6.2/lib/highline
/system_extensions.rb:36
            (root) at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/highline-1.6.2/lib/highline.rb:16
           require at org/jruby/RubyKernel.java:1038
           require at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/highline-1.6.2/lib/highline.rb:36
            (root) at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/highline-1.6.2/lib/highline/import.rb:10
           require at org/jruby/RubyKernel.java:1038
           require at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/highline-1.6.2/lib/highline/import.rb:36
            (root) at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/buildr-1.4.6-java/lib/buildr.rb:28
           require at org/jruby/RubyKernel.java:1038
           require at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/buildr-1.4.6-java/lib/buildr.rb:36
            (root) at /home/alexis/buildr-1.4.6/lib/ruby/gems/1.8/gems/buildr-1.4.6-java/bin/buildr:18
              load at org/jruby/RubyKernel.java:1063
            (root) at /home/alexis/buildr-1.4.6/bin/_buildr:19
```

Problems installing RJB

Most of the time issues related to installing rjb are a result of not having the JAVA_HOME environment variable set.

Windows

If you get something like the following:

Install required dependency rjb? [Yn] Y
Select which gem to install for your platform (i386-mswin32)
1. rjb 1.1.2 (ruby)
2. rjb 1.1.2 (x86-mswin32-60)
3. Skip this gem
4. Cancel installation
> 1
Building native extensions. This could take a while...
ERROR: While executing gem ... (Gem::Installer::ExtensionBuildError)
ERROR: Failed to build gem native extension.

ruby extconf.rb install buildr
checking for jni.h... no

Then choose #2

Mac OS X Leopard

Look here for a solution to installing rjb 1.1.x on Leopard.

Can't create Java VM

If the buildr or its install process fails with "can't create Java VM" message, one of the possible causes is that you're mixing 32-bit ruby and 64-bit java (or the reverse).

Running out of heap space

In the build itself

You can give the JVM more heap space by setting the `JAVA_OPTS` environment variables. This environment variable provides arguments for staring the JVM. For example, to set the heap space to 1GB:

```
$ export "JAVA_OPTS=-Xms1g -Xmx1g"
$ buildr compile
```

If you're sharing the build with other developers, you'll want to specify these options in the Buildfile itself. You can set the environment variable within the Buildfile, but make sure to do so at the very top of the Buildfile.

For example:

```
ENV['JAVA_OPTS'] = '-Xms1g -Xmx1g'
```

In tests

If you run tests in a separate VM (which is the default for most test frameworks in buildr), `JAVA_OPTS` will be ignored by the test runner. You can pass VM arguments to a forked test runner using the `:java_args` option for the test framework. E.g.,

```
test.using :java_args => [ '-Xmx1g' ]
```

Check the documentation for the test framework you are using to make sure it supports this.

RJB fails to compile

On Linux, BSD and Cygwin, RJB locates the JDK headers files – which it uses to compile a native C extension – based on the `JAVA_HOME` environment variable. Make sure `JAVA_HOME` points to the JDK, not JRE.

If you are using `sudo gem install`, note that some environments do not pass the `JAVA_HOME` environment variable over to `sudo`. To get around this, run `gem` with the `env JAVA_HOME=$JAVA_HOME` option:

```
$ sudo env JAVA_HOME=$JAVA_HOME gem install buildr
```

Segmentation Fault when running Java code

This is most likely a JVM inconsistency, for example, when part of the RJB library uses JDK 1.6, the other part uses JDK 1.5.

During installation RJB builds a native C extension using header files supplied by the JVM, and compiles a Java bridge class using the Javac. It is possible for RJB to use two different versions of the JVM, for example, if `JAVA_HOME` points to JDK 1.5, but `/usr/bin/javac` points to JDK 1.6.

Make sure `JAVA_HOME` and `/usr/bin/javac` both point to the same JDK:

```
echo $JAVA_HOME
ls -l /usr/bin/javac
```

Note: It seems that RJB works with Java 6, except when it doesn't, and for a few people it doesn't. For example, on OS X the default Ruby is 32-bit, but Java 6 is only available as 64-bit, and you can't mix architectures. In that case, either switch to Java 1.5, or simply run Buildr on JRuby using Java 6.

Bugs resulting from a dangling comma or period

Ruby statements don't need a delimiter and can span multiple lines, which can lead to bugs resulting from dangling commas and periods left at the end of the line. For example:

```
compile.with 'org.apache.axis2:axis2:jar:1.2',
test.with 'log4j:log4j:jar:1.1'
```

This is actually a single method call with two arguments, separated by a comma. The second argument is the result of calling `test.with`, and makes the test task a pre-requisite of the compile task, leading to a circular dependency.

As you can imagine this happens usually after editing, specifically for commas and periods which are small enough that you won't notice them from a cursory glance at the code, so if all else fails, search for lines that end with one of these characters.

Missing POM breaks transitive dependencies

Occasionally, artifacts are deployed to remote repositories with missing or broken POMs. Buildr will fail when attempting to resolve transitive dependencies with broken or missing POMs.

In this particular case, failing is doing the right thing. There's no way for Buildr to tell whether the POM is nowhere to be found, or just a temporary problem accessing the remote server.

If you can determine that the POM file is missing you can work around the problem in three ways. If you published the artifact, make the release again, getting it to upload the missing files.

If the source repository is not under your control, but you are also using your own repository for the project, you can always create a dummy POM in your own repository. Buildr will attempt to download the file from either repository, using the first file it finds.

Alternatively, you can make Buildr create a dummy POM file in the local repository, instead of downloading it from a remote repository. This example creates a dummy POM for Axis JAX-RPC:

```
artifact 'axis:axis-jaxrpc:pom:1.3' do |task|
  write task.name, <<-POM
    <project>
      <modelVersion>4.0.0</modelVersion>
      <groupId>axis</groupId>
      <artifactId>axis-jaxrpc</artifactId>
      <version>1.4</version>
    </project>
  POM
end
```

Buildr fails to run after install with a "stack level too deep (SystemStackError)" error

A particular quirk of an existing Ruby setup can cause problems when running Buildr. If a system already has several Ruby directories that are in the `PATH`, it is often nice (appropriate?) to have them in `RUBYLIB` as well (to be able to require them). If there are several of them a user may decide that `RUBYLIB=$PATH` is a good way to handle this (or some less automated method that has the same effect).

The culprit is having the Gem's binary directory show up in `RUBYLIB`. For example, Buildr's `bin/buildr` includes this line:

```
require 'buildr'
```

Under normal circumstances, this tells RubyGems to load `buildr.rb` from the Gem's library directory. When `RUBYLIB` points to the Gem's `bin` directory, it ends up loading itself repeatedly.

To solve this, remove Buildr's `bin` directory from `RUBYLIB`. Removing all directories that you don't actually need is better (other Gems may have the same problem).